RMT
Internet-Draft
Expires: June 10, 2004

T. Paila
Nokia
M. Luby
Digital Fountain
R. Lehtonen
TeliaSonera
V. Roca
INRIA Rhone-Alpes
R. Walsh
Nokia
December 11, 2003

FLUTE - File Delivery over Unidirectional Transport
draft-ietf-rmt-flute-07.txt

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http:// www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on June 10, 2004.

Copyright Notice

Copyright (C) The Internet Society (2003). All Rights Reserved.

Abstract

This document defines FLUTE, a protocol for the unidirectional delivery of files over the Internet, which is particularly suited to multicast networks. The specification builds on Asynchronous Layered Coding, the base protocol designed for massively scalable multicast distribution.

Table of Contents

1. Introduction

This document defines FLUTE version 1, a protocol for unidirectional delivery of files over the Internet. The specification builds on Asynchronous Layered Coding (ALC), version 1 [2], the base protocol designed for massively scalable multicast distribution. ALC defines transport of arbitrary binary objects. For file delivery applications mere transport of objects is not enough, however. The end systems need to know what do the objects actually represent. This document specifies a technique called FLUTE - a mechanism for signaling and mapping the properties of files to concepts of ALC in a way that allows receivers to assign those parameters for received objects. Consequently, throughout this document the term 'file' relates to an 'object' as discussed in ALC. Although this specification frequently makes use of multicast addressing as an example, the techniques are

similarly applicable for use with unicast addressing.

This document defines a specific transport application of ALC, adding the following specifications:

- Definition of a file delivery session built on top of ALC, including transport details and timing constraints.
- In-band signalling of the transport parameters of the ALC session.
- In-band signalling of the properties of delivered files.
- Details associated with the multiplexing of multiple files within a session.

This specification is structured as follows. Section 3 begins by defining the concept of the file delivery session. Following that it introduces the File Delivery Table that forms the core part of this specification. Further, it discusses multiplexing issues of transport objects within a file delivery session. Section 4 describes the use of congestion control and channels with FLUTE. Section 5 defines how the FEC Object Transmission Information is to be delivered within a file delivery session. Section 6 defines the required parameters for describing file delivery sessions in a general case. Section 7 outlines security considerations regarding file delivery with FLUTE. Last, there are two informative appendixes. The first appendix gives an example of File Delivery Table. The second appendix describes an envisioned receiver operation for the receiver of the file delivery session.

Statement of Intent

This memo contains part of the definitions necessary to fully specify a Reliable Multicast Transport protocol in accordance with RFC2357. As per RFC2357, the use of any reliable multicast protocol in the Internet requires an adequate congestion control scheme.

While waiting for such a scheme to be available, or for an existing scheme to be proven adequate, the Reliable Multicast Transport working group (RMT) publishes this Request for Comments in the "Experimental" category.

It is the intent of RMT to re-submit this specification as an IETF Proposed Standard as soon as the above condition is met.

1.1 Applicability Statement

1.1.1 The Target Application Space

FLUTE is applicable to the delivery of large and small files to many hosts, using delivery sessions of several seconds or more. For

instance, FLUTE could be used for the delivery of large software updates to many hosts simultaneously. It could also be used for continuous, but segmented, data such as time-lined text for subtitling - potentially leveraging its layering inheritance from ALC and LCT to scale the richness of the session to the congestion status of the network. It is also suitable for the basic transport of metadata, for example SDP files which enable user applications to access multimedia sessions.

1.1.2 The Target Scale

Massive scalability is a primary design goal for FLUTE. IP multicast is inherently massively scalable, but the best effort service that it provides does not provide session management functionality, congestion control or reliability. FLUTE provides all of this using ALC and IP multicast without sacrificing any of the inherent scalability of IP multicast.

1.1.3 Intended Environments

All of the environmental requirements and considerations that apply to the ALC building block [2] and to any additional building blocks that FLUTE uses also apply to FLUTE.

FLUTE can be used with both multicast and unicast delivery, but it's

primary application is for unidirectional multicast delivery. FLUTE requires connectivity between a sender and receivers but does not require connectivity from receivers to a sender. FLUTE inherently works with all types of networks, including LANs, WANs, Intranets, the Internet, asymmetric networks, wireless networks, and satellite networks. Thus, the inherent raw scalability of FLUTE is unlimited.

FLUTE is compatible with both IPv4 or IPv6 as no part of the packet is IP version specific. FLUTE works with both multicast models: Any-Source Multicast (ASM) [12] and the Source-Specific Multicast (SSM) [14].

FLUTE is applicable for both Internet use, with a suitable congestion control building block, and provisioned/controlled systems, such as delivery over wireless broadcast radio systems.

1.1.4 Weaknesses

Some networks are not amenable to some congestion control protocols that could be used with FLUTE. In particular, for a satellite or wireless network, there may be no mechanism for receivers to effectively reduce their reception rate since there may be a fixed transmission rate allocated to the session.

FLUTE provides reliability using the FEC building block. This will reduce the error rate as seen by applications. However, FLUTE does not provide a method for senders to verify the reception success of receivers, and the specification of such a method is outside the scope of this document.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [1].

The terms "object" and "transport object" are consistent with the definitions in ALC [2] and LCT [3]. The terms "file" and "source object" are pseudonyms for "object".

3. File delivery

Asynchronous Layered Coding [2] is a protocol designed for delivery of arbitrary binary objects. It is especially suitable for massively scalable, unidirectional, multicast distribution. ALC provides the basic transport for FLUTE, and thus FLUTE inherits the requirements of ALC.

This specification is designed for the delivery of files. The core of this specification is to define how the properties of the files are carried in-band together with the delivered files.

As an example, let us consider a 5200 byte file referred to by "www.ex.com/docs/file.txt". Using the example, the following properties describe the properties that need to be conveyed by the file delivery protocol.

* Globally unique identifier of the file, expressed as either absolute or relative URI. This is used as an identifier and optionally also as a location for the file. In the above example: "www.ex.com/docs/file.txt".
* File name (usually, this can be concluded from the URI). In the above example: "file.txt".
* File type, expressed as MIME media type (usually, this can also be concluded from the extension of the file name). In the above example: "text/plain". If an explicit value for the MIME type is provided separately from the file extension and does not match the MIME type of the file extension then the explicitly provided value MUST be used as the MIME type.
* File size, expressed in bytes. In the above example: "5200". If the file is content encoded then this is the file size before content encoding.
* Content encoding of the file, within transport. In the above example, the file could be encoded using ZLIB [10]. In this case the size of the transport object carrying the file would probably differ from the file size. The transport object size is delivered to receivers as part of the FLUTE protocol.
* Security properties of the file such as digital signatures, message digests, etc. For example, one could use SMIME [17] as the content encoding type for files with this authentication wrapper, and one could use XML-DSIG [18] to digitally sign an FDT Instance.

3.1 File delivery session

ALC is a protocol instantiation of Layered Coding Transport building block (LCT) [3]. Thus ALC inherits the session concept of LCT. In this document we will use the concept ALC/LCT session to collectively denote the interchangeable terms ALC session and LCT session.

An ALC/LCT session consists of a set of logically grouped ALC/LCT

channels associated with a single sender sending packets with ALC/LCT headers for one or more objects. An ALC/LCT channel is defined by the combination of a sender and an address associated with the channel by the sender. A receiver joins a channel to start receiving the data packets sent to the channel by the sender, and a receiver leaves a channel to stop receiving data packets from the channel.

One of the fields carried in the ALC/LCT header is the Transport Session Identifier (TSI). The TSI is scoped by the source IP address, and the (source IP address, TSI) pair uniquely identifies a session, i.e., the receiver uses this pair carried in each packet to uniquely identify from which session the packet was received. In case multiple objects are carried within a session another field within the ALC/LCT header, the Transport Object Identifier (TOI), identifies from which object within the session the data in the packet was generated. Note that each object is associated with a unique TOI within the scope of a session.

When FLUTE is used for file delivery over ALC the following rules apply:

* The ALC/LCT session is called file delivery session.
* The ALC/LCT concept of 'object' denotes either a 'file' or a 'File Delivery Table Instance' (section 3.2)
* The TOI field MUST be included in ALC packets sent within a FLUTE session, with the exception that ALC packets sent in a FLUTE session with the Close Session (A) flag set to 1 (signaling the end of the session) and containing no payload MAY omit the TOI. See Section 5.1 of RFC 3451 [3] for the LCT definition of the Close Session flag, and see Section 4.2 of RFC 3450 [2] for an example of its use within an ALC packet.
* The TOI value '0' is reserved for delivery of File Delivery Table Instances. Each File Delivery Table Instance is uniquely identified by an FDT Instance ID.
* Each file in a file delivery session MUST be associated with a TOI (>0) in the scope of that session.
* Information carried in the headers and the payload of a packet is scoped by the source IP address and the TSI. Information particular to the object carried in the headers and the payload of a packet is further scoped by the TOI for file objects, and is further scoped by both the TOI and the FDT Instance ID for FDT Instance objects.

3.2 File Delivery Table

The File Delivery Table (FDT) provides a means to describe various attributes associated with files that are to be delivered within the file delivery session. The following lists are examples of such attributes, and are not intended to be mutually exclusive nor exhaustive.

Attributes related to the delivery of file:

- TOI value that represents the file
- FEC Instance ID
- FEC Object Transmission Information
- Size of the transport object carrying the file
- Aggregate rate of sending packets to all channels

Attributes related to the file itself:

- Name, Identification and Location of file (specified by the URI)
- MIME media type of file
- Size of file
- Encoding of file
- Message digest of file

Some of these attributes MUST be included in the file description entry for a file, others are optional, as defined in section 3.4.2.

Logically, the FDT is a set of file description entries for files to be delivered in the session. Each file description entry MUST include the TOI for the file that it describes and the URI indicating the location of the file. The TOI is included in each ALC/LCT data packet during the delivery of the file, and thus the TOI carried in the file description entry is how the receiver determines which ALC/LCT data packets contain information about which file. Each file description entry may also contain one or more descriptors that map the above-mentioned attributes to the file.

Each file delivery session MUST have an FDT that is local to the given session. The FDT MUST provide a file description entry mapped to a TOI for each file appearing within the session. An object that

is delivered within the ALC session, but not described in the FDT, is not considered a 'file' belonging to the file delivery session. Handling of these unmapped TOIs (TOIs that are not resolved by the FDT) is out of scope of this specification.

Within the file delivery session the FDT is delivered as FDT Instances. An FDT Instance contains one or more file description entries of the FDT. Any FDT Instance can be equal to, a subset of, a superset of, or complement any other FDT Instance. A certain FDT Instance may be repeated several times during a session, even after subsequent FDT Instances (with higher FDT Instance ID numbers) have been transmitted. Each FDT Instance contains at least a single file description entry and at most the complete FDT of the file delivery session.

A receiver of the file delivery session keeps an FDT database for received file description entries. The receiver maintains the database, for example, upon reception of FDT Instances. Thus, at any given time the contents of the FDT database represent the receiver's current view of the FDT of the file delivery session. Since each receiver behaves independently of other receivers, it SHOULD NOT be assumed that the contents of the FDT database are the same for all the receivers of a given file delivery session.

Since FDT database is an abstract concept, the structure and the maintaining of the FDT database are left to individual implementations and are thus out of scope of this specification.

3.3 Dynamics of FDT Instances within file delivery session

The following rules define the dynamics of the FDT Instances within a file delivery session:

* For every file delivered within a file delivery session there MUST be a file description entry included in at least one FDT Instance sent within the session. A file description entry contains at a minimum the mapping between the TOI and the URI.

* An FDT Instance MAY appear in any part of the file delivery session and packets for an FDT Instance MAY be interleaved with packets for other files or other FDT Instances within a session.

* The TOI value of '0' MUST be reserved for delivery of FDT Instances. The use of other TOI values for FDT Instances is outside the scope of this specification.

* FDT Instance is identified by the use of a new fixed length LCT Header Extension EXT_FDT (defined later in this section). Each FDT Instance is uniquely identified within the file delivery session by its FDT Instance ID. Any ALC/LCT packet carrying FDT Instance (indicated by TOI = 0) MUST include EXT_FDT.

* It is RECOMMENDED that FDT Instance that contains the file

description entry for a file is sent prior to the sending of the described file within a file delivery session.

* Within a file delivery session, any TOI MAY be described more than once. An example: previous FDT Instance 0 describes TOI of value '3'. Now, subsequent FDT Instances can either keep TOI '3' unmodified on the table, not to include it or complement the description. However, subsequent FDT Instances MUST NOT change the parameters already described for a specific TOI.

* An FDT Instance is valid until its expiration time. The expiration time is expressed within the FDT Instance payload as a 32 bit data field. The value of the data field represents the 32 most significant bits of a 64 bit Network Time Protocol (NTP) [5] time value. These 32 bits provide an unsigned integer representing the time in seconds relative to 0 hours 1 January 1900. Handling of wrap-around of the 32 bit time is outside the scope of NTP and FLUTE.

* The receiver SHOULD NOT use a received FDT Instance to interpret packets received beyond the expiration time of the FDT Instance.

* A sender MUST use an expiry time in the future upon creation of an FDT Instance.

* Any FEC Encoding ID MAY be used for the sending of FDT Instances. The default is to use FEC Encoding ID 0 for the sending of FDT Instances. (Note that since FEC Encoding ID 0 is the default for FLUTE, this implies that Source Block Number and Encoding Symbol ID lengths both default to 16 bytes each.)

Generally, a receiver needs to receive an FDT Instance describing a file before it is able to recover the file itself. In this sense FDT Instances are of higher priority than files. Thus, it is RECOMMENDED that FDT Instances describing a file be sent with at least as much reliability within a session (more often or with more FEC protection) as the files they describe. In particular, if FDT Instances are generally longer than one Encoding Symbol in length it is RECOMMENDED that an FEC code that can provide protection against loss be used for delivering FDT Instances, i.e., in this case it is RECOMMENDED not to use FEC Encoding ID 0. How often the description of a file is sent in

an FDT Instance or how much FEC protection is provided for each FDT Instance (if the FDT Instance is longer than one Encoding Symbol) is dependent on the particular application and outside the scope of this document.

3.4 Structure of FDT Instance packets

FDT Instances are carried in ALC packets with TOI = 0 and with an additional REQUIRED LCT Header extension called the FDT Instance Header. The FDT Instance Header (EXT_FDT) contains the FDT Instance ID that uniquely identifies FDT Instances within a file delivery session. The FDT Instance Header is placed in the same way as any other LCT extension header. There MAY be other LCT extension headers in use.

The LCT extension headers are followed by the FEC Payload ID, and finally the Encoding Symbols for the FDT Instance which contains one or more file description entries. The FDT Instance MAY span over several ALC packets - the number of ALC packets is a function of the FEC Object Transmission Information associated to this FDT Instance. The FDT Instance Header is carried in each ALC packet carrying the FDT Instance. The FDT Instance Header is identical for all ALC/LCT packets for a particular FDT Instance.

The overall format of ALC/LCT packets carrying an FDT Instance is depicted in the Figure 1 below. All integer fields are carried in "big-endian" or "network order" format, that is, most significant byte (octet) first. As defined in [2], all ALC/LCT packets are sent using UDP.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           UDP header                          |
|                                                               |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|              Default LCT header (with TOI = 0)                |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          LCT header extensions (EXT_FDT, EXT_FTI, etc.)       |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        FEC Payload ID                         |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                Encoding Symbol for FDT Instance               |
|                             ...                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```


Figure 1 - Overall FDT Packet

3.4.1 Format of FDT Instance Header

FDT Instance Header (EXT_FDT) is a new fixed length, ALC PI specific LCT header extension [3]. The Header Extension Type (HET) for the extension is 192. Its format is defined below:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   HET = 192   |   V   |          FDT Instance ID              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Version of FLUTE (V), 4 bits:

This document specifies FLUTE version 1. Hence in any ALC packet that carries FDT Instance and that belongs to the file delivery session as specified in this specification MUST set this field to '1'.

FDT Instance ID, 20 bits:

For each file delivery session the numbering of FDT Instances starts from '0' and is incremented by exactly one for each subsequent FDT Instance. After reaching the maximum value (2^{20}-1), the numbering starts again from '0'. When wraparound from 2^{20}-1 to 0 occurs, 0 is considered higher than 2^{20}-1. Receiver handling of wraparound and other special situations (for example, missing FDT Instance IDs resulting in longer increments than one) is left out of this specification to individual implementations of FLUTE.

3.4.2 Syntax of FDT Instance

The FDT Instance contains file description entries that provide the mapping functionality described in 3.2 above.

The FDT Instance is an XML structure that has a single root element "FDT-Instance". The "FDT-Instance" element MUST contain "Expires" attribute, which tells the expiry time of the FDT Instance. In addition, the "FDT-Instance" element MAY contain "Complete" attribute (boolean), which MAY be used to signal that the given FDT Instance is the last FDT Instance to be expected on this file delivery session. For each file to be declared in the given FDT Instance there is a single file description entry in the FDT Instance. Each entry is

represented by element "File" which is a child element of the FDT Instance structure.

The attributes of "File" element in the XML structure represent the attributes given to the file that is delivered in the file delivery session. Each "File" element MUST contain at least two attributes "TOI" and "Content-Location". "TOI" MUST be assigned a valid TOI value as described in section 3.3 above. "Content-Location" MUST be assigned a valid URI as defined in [6].

In addition to mandatory attributes, the "File" entity MAY contain other attributes of which the following are specifically pointed out.

* If the MIME type of the file is described, attribute "Content-Type" MUST be used for the purpose as defined in [6].

* If the length of the file is described, attribute "Content-Length" MUST be used for the purpose as defined in [6]. If the length of the file is different than the length of the transport object that carries it (the file was content encoded before transport), another attribute "Transfer-Length" MAY be used. The attribute "Transfer-Length" specifies the size of the transport object in bytes.

* If the content encoding scheme of the file is described, attribute "Content-Encoding" MUST be used for the purpose as defined in [6].

* If the MD5 message digest of the file is described, attribute "Content-MD5" MUST be used for the purpose as defined in [6].

* The FEC Object Transmission Information attributes as described in

section 5.2.

The following specifies the XML Schema [8][9] for FDT Instance:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema
xmlns:fl="http://www.example.com/flute"
elementFormDefault:xs="qualified"
targetNamespace:xs="http://www.example.com/flute">
  <xs:element name="FDT-Instance">
   <xs:complexType>
    <xs:attribute name="Expires" type="xs:string" use="required"/>
    <xs:attribute name="Complete" type="xs:boolean" use="optional"/>
    <xs:sequence>
     <xs:element name="File" maxOccurs="unbounded">
      <xs:complexType>
```



```
       <xs:attribute name="Content-Location"
                     type="xs:anyURI" use="required"/>
       <xs:attribute name="TOI"
                     type="xs:positiveInteger" use="required"/>
       <xs:attribute name="Content-Length"
                     type="xs:unsignedLong" use="optional"/>
       <xs:attribute name="Transfer-Length"
                     type="xs:unsignedLong" use="optional"/>
       <xs:attribute name="Content-Type"
                     type="xs:string" use="optional"/>
       <xs:attribute name="Content-Encoding"
                     type="xs:string" use="optional"/>
       <xs:attribute name="Content-MD5"
                     type="xs:base64Binary" use="optional"/>
       <xs:attribute name="FEC-OTI-FEC-Instance-ID"
                     type="xs:unsignedLong" use="optional"/>
       <xs:attribute name="FEC-OTI-Maximum-Source-Block-Length"
                     type="xs:unsignedLong" use="optional"/>
       <xs:attribute name="FEC-OTI-Encoding-Symbol-Length"
                     type="xs:unsignedLong" use="optional"/>
       <xs:attribute name="FEC-OTI-Max-Number-of-Encoding-Symbols"
                     type="xs:unsignedLong" use="optional"/>
       <xs:anyAttribute processContents="skip" />
      </xs:complexType>
     </xs:element>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
</xs:schema>
```

Any XML document that conforms with the above XML Schema is a valid FDT. This way FDT provides extensibility to support private attributes within the file description entries. Those could be, for example, the attributes related to the delivery of the file (timing, packet transmission rate, etc.).

In case the basic FDT XML Schema is extended in terms of new descriptors, those MUST be placed within the attributes of the element "File". It is RECOMMENDED that the new descriptors applied in

the FDT are in the format of MIME fields and are either defined in HTTP/1.1 specification [6] or otherwise well-known specification.

3.4.3 Content Encoding of FDT Instance

The FDT Instance itself MAY be content encoded, for example compressed. This specification defines FDT Instance Content Encoding Header (EXT_CENC). EXT_CENC is a new fixed length, ALC PI specific LCT header extension [3]. The Header Extension Type (HET) for the

extension is 193. If the FDT Instance is content encoded, the EXT_CENC MUST be used to signal the content encoding type. In that case, EXT_CENC header extension MUST be used in all ALC packets carrying the same FDT Instance ID. Consequently, when EXT_CENC header is used, it MUST be used together with a proper FDT Instance Header (EXT_FDT). Within a file delivery session, FDT Instances that are not content encoded and FDT Instances that are content encoded MAY both appear. If content encoding is not used for a given FDT Instance, the EXT_CENC MUST NOT be used in any packet carrying the FDT Instance. The format of EXT_CENC is defined below:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   HET = 193   |     CENC      |          Reserved             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Content Encoding Algoritm (CENC), 8 bits:

This field signals the content encoding algorithm used in the FDT Instance payload. The definition of this field is outside the scope of this specification. Applicable content encoding algorithms include, for example, ZLIB [10], DEFLATE [15] and GZIP [16].

Reserved, 16 bits:

This field MUST be set to all '0'.

3.5 Multiplexing of files within a file delivery session

The delivered files are carried as transport objects (identified with TOIs) in the file delivery session. All these objects, including the FDT Instances, MAY be multiplexed in any order and in parallel with each other within a session, i.e., packets for one file MAY be interleaved with packets for other files or other FDT Instances within a session.

Multiple FDT Instances MAY be delivered in a single session using TOI = 0. In this case, it is RECOMMENDED that the sending of a previous FDT Instance SHOULD end before the sending of the next FDT Instance starts. However, due to unexpected network conditions, packets for the FDT Instances MAY be interleaved. A receiver can determine which FDT Instance a packet contains information about since the FDT Instances are uniquely identified by their FDT Instance ID carried in the EXT_FDT headers.

4. Channels, congestion control and timing

ALC/LCT has a concept of channels and congestion control. There are four scenarios FLUTE is envisioned to be applied.

(a) Use a single channel and a single-rate congestion control protocol.

(b) Use multiple channels and a multiple-rate congestion control protocol. In this case the FDT Instances MAY be delivered on more than one channel.

(c) Use a single channel without congestion control supplied by ALC, but only when in a controlled network environment where flow/congestion control is being provided by other means.

(d) Use multiple channels without congestion control supplied by ALC, but only when in a controlled network environment where flow/congestion control is being provided by other means. In this case the FDT Instances MAY be delivered on more than one channel.

When using just one channel for a file delivery session, as in (a) and (c), the notion of 'prior' and 'after' are intuitively defined for the delivery of objects with respect to their delivery times.

However, if multiple channels are used, as in (b) and (d), it is not straightforward to state that an object was delivered 'prior' to the other. An object may begin to be delivered on one or more of those channels before the delivery of a second object begins. However, the use of multiple channels/layers may complete the delivery of the second object before the first. This is not a problem when objects are delivered sequentially using a single channel. Thus, if the application of FLUTE has a mandatory or critical requirement that the first transport object must complete 'prior' to the second one, it is RECOMMENDED that only a single channel is used for the file delivery session.

Furthermore, if multiple channels are used then a receiver joined to the session at a low reception rate will only be joined to the lower layers of the session. Thus, since the reception of FDT Instances is of higher priority than the reception of files (because the reception of files depends on the reception of an FDT Instance describing it), the following is RECOMMENDED:

1. The layers to which packets for FDT Instances are sent SHOULD NOT be biased towards those layers to which lower rate receivers are not joined. For example, it is ok to put all the packets for an FDT Instance into the lowest layer (if this layer carries enough

packets to deliver the FDT to higher rate receivers in a reasonable amount of time), but it is not ok to put all the packets for an FDT Instance into the higher layers that only high rate receivers will receive.

2. If FDT Instances are generally longer than one Encoding Symbol in length and some packets for FDT Instances are sent to layers that lower rate receivers do not receive, an FEC Encoding other than FEC Encoding ID 0 SHOULD be used to deliver FDT Instances. This is because in this case, even when there is no packet loss in the network, a lower rate receiver will not receive all packets sent for an FDT Instance.

5. Delivering FEC Object Transmission Information

FLUTE inherits the use of FEC building block [4] from ALC. When using FLUTE for file delivery over ALC the FEC Object Transmission Information MUST be delivered in-band within the file delivery session. In this section, two methods are specified for FLUTE for this purpose: the use of ALC specific LCT extension header EXT_FTI [2], and, the use of FDT.

The receiver of file delivery session MUST support delivery of FEC Object Transmission Information using the EXT_FTI for the FDT Instances carried using TOI value 0. For the TOI values other than 0 the receiver MUST support both methods: the use of EXT_FTI and the use of FDT.

The FEC Object Transmission Information that needs to be delivered to receivers MUST be exactly the same whether it is delivered using EXT_FTI or using FDT (or both). Section 5.1 describes the required FEC Object Transmission Information that MUST be delivered to receivers for various FEC Encoding IDs. In addition, it describes the delivery using EXT_FTI. Section 5.2 describes the delivery using FDT.

The FEC Object Transmission Information regarding a given TOI may be available from several sources. In this case, it is RECOMMENDED that the receiver of the file delivery session prioritizes the sources in the following way (in the order of decreasing priority).

1. FEC Object Transmission Information that is available in EXT_FTI.

2. FEC Object Transmission Information that is available in the FDT.

5.1 Use of EXT_FTI for delivery of FEC Object Transmission Information

As specified in [2], the EXT_FTI header extension is intended to carry in band the FEC Object Transmission Information for an object. It is left up to individual implementations to decide how frequently and in which ALC packets the EXT_FTI header extension is included. In environments with higher packet loss rate, the EXT_FTI might need to

be included more frequently in ALC packets than in environments with low error probability. The EXT_FTI MUST be included in at least one sent ALC packet for each FDT Instance.

The ALC specification does not define the format or the processing of the EXT_FTI header extension. The following sections specify EXT_FTI when used in FLUTE.

In FLUTE, the FEC Encoding ID (8 bits) is carried in the Codepoint field of the ALC/LCT header.

5.1.1 General EXT_FTI format

The general EXT_FTI format specifies the structure and those attributes of FEC Object Transmission Information that are applicable to any FEC Encoding ID.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   HET = 64    |      HEL      |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
|                        Transfer Length                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   FEC Instance ID             | FEC Enc. ID Specific Format   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Header Extension Type (HET), 8 bits:

64 as defined in [2]

Header Extension Length (HEL), 8 bits:

The length of the whole Header Extension field, expressed in multiples of 32-bit words. This length includes the FEC Encoding ID specific format part.

Transfer Length, 48 bits:

The length of the transport object that carries the file in bytes.

(This is the same as the file length if the file is not content encoded.)

FEC Instance ID, optional, 16 bits:

This field is used for FEC Instance ID. It is only present if the value of FEC Encoding ID is in the range of 128-255. When the value of FEC Encoding ID is in the range of 0-127, this field is set to 0.

FEC Encoding ID Specific Format:

Different FEC encoding schemes will need different sets of encoding parameters. Thus, the structure and length of this field depends on FEC Encoding ID. The next sections specify structure of this field for FEC Encoding ID numbers 0, 128, 129 and 130.

5.1.2 FEC Encoding ID specific formats for EXT_FTI

5.1.2.1 FEC Encoding IDs 0, 128, and 130

FEC Encoding ID 0 is 'Compact No-Code FEC' (Fully-Specified) [7]. FEC Encoding ID 128 is 'Small Block, Large Block and Expandable FEC' (Under-Specified) [4]. FEC Encoding ID 130 is 'Compact FEC' (Under-Specified) [7]. For these FEC Encoding IDs, the FEC Encoding ID specific format of EXT_FTI is defined as follows.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
                                +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   General EXT_FTI format       |     Encoding Symbol Length    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Maximum Source Block Length                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Encoding Symbol Length, 16 bits:

Length of Encoding Symbol in bytes.

All Encoding Symbols of a transport object MUST be equal to this length, with the optional exception of the last source symbol of the last source block (so that redundant padding is not mandatory in this last symbol). This last source symbol MUST be logically padded out with zeroes when another Encoding Symbol is computed based on this source symbol to ensure the same interpretation of this Encoding Symbol value by the sender and receiver. However, this padding need not be actually sent with the data of the last source symbol.

Maximum Source Block Length, 32 bits

The maximum number of source symbols per source block.

This EXT_FTI specification requires that an algorithm is known to both sender and receivers for determining the size of all source blocks of the transport object that carries the file identified by the TOI (or within the FDT Instance identified by the TOI and the FDT Instance ID). The algorithm SHOULD be the same for all files using the same FEC Encoding ID within a session.

Section 5.1.2.3 describes an algorithm that is RECOMMENDED for this use.

For the FEC Encoding IDs 0, 128 and 130, this algorithm is the only well known way the receiver can determine the length of each source block. Thus, the algorithm does two things: (a) it tells the receiver the length of each particular source block as it is receiving packets for that source block - this is essential to all of these FEC schemes; and, (b) it provides the source block structure immediately to the receiver so that the receiver can determine where to save recovered source blocks at the beginning - this is an optimization which is essential for some implementations.

5.1.2.2 FEC Encoding ID 129

Small Block Systematic FEC (Under-Specified). The FEC Encoding ID specific format of EXT_FTI is defined as follows.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
                                +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   General EXT_FTI format       |     Encoding Symbol Length    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Maximum Source Block Length  | Max. Num. of Encoding Symbols |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Encoding Symbol Length, 16 bits:

Length of Encoding Symbol in bytes.

Maximum Source Block Length, 16 bits:

The maximum number of source symbols per source block.

Maximum Number of Encoding Symbols, 16 bits:

Maximum number of Encoding Symbols that can be generated for a source block.

All Encoding Symbols of a transport object MUST be equal to this length, with the optional exception of the last source symbol of the last source block (so that redundant padding is not mandatory in this last symbol). This last source symbol MUST be logically padded out with zeroes when another Encoding Symbol is computed based on this source symbol to ensure the same interpretation of this Encoding Symbol value by the sender and receiver. However, this padding need not be actually sent with the data of the last source symbol.

This EXT_FTI specification requires that an algorithm is known to both sender and receivers for determining the size of all source blocks of the transport object that carries the file identified by the TOI (or within the FDT Instance identified by the TOI and the FDT Instance ID). The algorithm SHOULD be the same for all files using the same FEC Encoding ID within a session.

Section 5.1.2.3 describes an algorithm that is RECOMMENDED for this use. For FEC Encoding ID 129 the FEC Payload ID in each data packet already contains the source block length for the source block corresponding to the Encoding Symbol carried in the data packet. Thus, the algorithm for computing source blocks for FEC Encoding ID 129 could be to just use the source block lengths carried in data packets within the FEC Payload ID. However, the algorithm described in Section 5.1.2.3 is useful for the receiver to compute the source block structure at the beginning of the reception of data packets for the file. If the algorithm described in Section 5.1.2.3 is used then it MUST be the case that the source block lengths that appear in data packets agree with the source block lengths calculated by the algorithm.

5.1.2.3 Algorithm for Computing Source Block Structure

This algorithm computes a source block structure so that all source blocks are as close to being equal length as possible. A first number of source blocks are of the same larger length, and the remaining second number of source blocks are sent of the same smaller length. The total number of source blocks (N), the first number of source blocks (I), the second number of source blocks (N-I), the larger length (A_large) and the smaller length (A_small) are calculated thus,

Input:

B -- Maximum Source Block Length, i.e., the maximum number of source symbols per source block

L -- Transfer Length in bytes

E -- Encoding Symbol Length in bytes

Output:

N -- The number of source blocks into which the transport object is partitioned.

The number and lengths of source symbols in each of the N source blocks.

Algorithm:

(a) The number of source symbols in the transport object is computed as T = L/E rounded up to the nearest integer.

(b) The transport object is partitioned into N source blocks, where N = T/B rounded up to the nearest integer

(c) The average length of a source block, A = T/N (this may be non-integer)

(d) A_large = A rounded up to the nearest integer (it will always be the case that the value of A_large is at most B)

(e) A_small = A rounded down to the nearest integer (if A is an integer A_small = A_large, and otherwise A_small = A_large - 1)

(f) The fractional part of A, A_fraction = A - A_small

(g) I = A_fraction * N (I is an integer between 0 and N-1)

(h) Each of the first I source blocks consists of A_large source symbols, each source symbol is E bytes in length. Each of the remaining N-I source blocks consist of A_small source symbols, each source symbol is E bytes in length except that the last source symbol of the last source block is L-(((L-1)/E) rounded down to the nearest integer)*E bytes in length.

5.2 Use of FDT for delivery of FEC Object Transmission Information

The FDT delivers FEC Object Transmission Information for each file using an appropriate attribute within the "File" element of the FDT structure. For future FEC Encoding IDs, if the attributes listed below do not fulfil the needs of describing the FEC Object Transmission Information then additional new attributes MAY be used.

* "Transfer-Length" is semantically equivalent with the field

"Transfer Length" of EXT_FTI.

* "FEC-OTI-FEC-Instance-ID" is semantically equivalent with the field "FEC Instance ID" of EXT_FTI.

* "FEC-OTI-Maximum-Source-Block-Length" is semantically equivalent with the field "Maximum Source Block Length" of EXT_FTI for FEC Encoding IDs 0, 128 and 130, and semantically equivalent with the field "Maximum Source Block Length" of EXT_FTI for FEC Encoding ID

129.

* "FEC-OTI-Encoding-Symbol-Length" is semantically equivalent with the field "Encoding Symbol Length" of EXT_FTI for FEC Encoding IDs 0, 128, 129 and 130.

* "FEC-OTI-Max-Number-of-Encoding-Symbols" is semantically equivalent with the field "Maximum Number of Encoding Symbols" of EXT_FTI for FEC Encoding ID 129.

6. Describing file delivery sessions

To start receiving a file delivery session, the receiver needs to know transport parameters associated with the session. Interpreting these parameters and starting the reception therefore represents the entry point from which thereafter the receiver operation falls into the scope of this specification. According to [2], the transport parameters of an ALC/LCT session that the receiver needs to know are:

* The source IP address;

* The number of channels in the session;

* The destination IP address and port number for each channel in the session;

* The Transport Session Identifier (TSI) of the session;

* An indication that the session carries packets for more than one object, and this indicates that the TOI field is included in sent packets;

Optionally, the following parameters MAY be associated with the session (Note, the list is not exhaustive):

* The start time and end time of the session;

* FEC Encoding ID and FEC Instance ID when the default FEC Encoding ID 0 is not used for the delivery of FDT;

* Content Encoding format if optional content encoding of FDT Instance is used, e.g., compression;

* Some information that tells receiver, in the first place, that the session contains files that are of interest

How the receiver acquires the above-mentioned parameters is out of

scope of this document. The specification, in particular, does not mandate or exclude any mechanism. The description can be conveyed to the receiver via techniques such as Session Announcement Protocol [11], email, accessing URL, manual configuration, etc. Similarly the format of this session description is out of the scope of this document.

7. Security Considerations

The same security consideration that apply to ALC and to the LCT, FEC and the congestion control building block used in conjunction with FLUTE also apply to FLUTE.

Because of the use of FEC, FLUTE is especially vulnerable to denial-of-service attacks by attackers that try to send forged packets to the session which would prevent successful reconstruction or cause inaccurate reconstruction of large portions of the FDT or file by receivers. Like ALC, FLUTE is particularly affected by such an attack because many receivers may receive the same forged packet. A malicious attacker may spoof file packets and cause incorrect recovery of a file.

Even more damaging, a malicious forger may spoof FDT Instance packets, for example sending packets with erroneous FDT-Instance fields. Many attacks can follow this approach. For instance a malicious attacker may alter the Content-Location field of TOI 'n', to make it point to a system file or a user configuration file. Then, TOI 'n' can carry a Trojan horse or some other type of virus. It is thus RECOMMENDED that the FLUTE delivery service at the receiver does not have write access to the system files or directories, or any other critical areas. Another example is generating a bad Content-MD5 sum, leading receivers to reject the associated file that will be declared corrupted. The Content-Encoding can also be modified, which also prevents the receivers to correctly handle the associated file. These examples show that the FDT information is critical to the FLUTE delivery service.

At the application level, it is RECOMMENDED that an integrity check on the entire received object be done once the object is reconstructed to ensure it is the same as the sent object, especially for objects that are FDT Instances. Moreover, in order to obtain strong cryptographic integrity protection a digital signature verifiable by the receiver SHOULD be used to provide this application level integrity check. However, if even one corrupted or forged packet is used to reconstruct the object, it is likely that the received object will be reconstructed incorrectly. This will appropriately cause the integrity check to fail and in this case the inaccurately reconstructed object SHOULD be discarded. Thus, the

acceptance of a single forged packet can be an effective denial of service attack for distributing objects, but an object integrity check at least prevents inadvertent use of inaccurately reconstructed objects. The specification of an application level integrity check of the received object is outside the scope of this document.

At the packet level, it is RECOMMENDED that a packet level authentication be used to ensure that each received packet is an authentic and uncorrupted packet containing FEC data for the object arriving from the specified sender. Packet level authentication has the advantage that corrupt or forged packets can be discarded individually and the received authenticated packets can be used to accurately reconstruct the object. Thus, the effect of a denial of service attack that injects forged packets is proportional only to the number of forged packets, and not to the object size. Although there is currently no IETF standard that specifies how to do multicast packet level authentication, TESLA [13] is a known multicast packet authentication scheme that would work.

In addition to providing protection against reconstruction of inaccurate objects, packet level authentication can also provide some protection against denial of service attacks on the multiple rate congestion control. Attackers can try to inject forged packets with incorrect congestion control information into the multicast stream, thereby potentially adversely affecting network elements and receivers downstream of the attack, and much less significantly the rest of the network and other receivers. Thus, it is also RECOMMENDED that packet level authentication be used to protect against such attacks. TESLA [13] can also be used to some extent to limit the damage caused by such attacks. However, with TESLA a receiver can only determine if a packet is authentic several seconds after it is received, and thus an attack against the congestion control protocol can be effective for several seconds before the receiver can react to slow down the session reception rate.

Reverse Path Forwarding checks SHOULD be enabled in all network routers and switches along the path from the sender to receivers to limit the possibility of a bad agent injecting forged packets into the multicast tree data path.

A receiver with an incorrect or corrupted implementation of the multiple rate congestion control building block may affect health of the network in the path between the sender and the receiver, and may also affect the reception rates of other receivers joined to the session. It is therefore RECOMMENDED that receivers be required to identify themselves as legitimate before they receive the Session Description needed to join the session. How receivers identify themselves as legitimate is outside the scope of this document.

Another vulnerability of FLUTE is the potential of receivers obtaining an incorrect Session Description for the session. The consequences of this could be that legitimate receivers with the wrong Session Description are unable to correctly receive the session content, or that receivers inadvertently try to receive at a much higher rate than they are capable of, thereby disrupting traffic in portions of the network. To avoid these problems, it is RECOMMENDED

that measures be taken to prevent receivers from accepting incorrect Session Descriptions, e.g., by using source authentication to ensure that receivers only accept legitimate Session Descriptions from authorized senders. How this is done is outside the scope of this document.

8. IANA Considerations

No information in this specification is directly subject to IANA registration. However, building blocks components used by ALC may introduce additional IANA considerations. In particular, the FEC building block used by FLUTE does require IANA registration of the FEC codecs used.

9. Acknowledgements

The following persons have contributed to this specification: Brian Adamson, Mark Handley, Esa Jalonen, Roger Kermode, Juha-Pekka Luoma, Jani Peltotalo, Sami Peltotalo, Topi Pohjolainen and Lorenzo Vicisano. The authors would like to thank all the contributors for their valuable work in reviewing and providing feedback regarding this specification.

Normative references

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, BCP 14, March 1997.

[2] Luby, M., Gemmell, J., Vicisano, L., Rizzo, L. and J. Crowcroft, "Asynchronous Layered Coding (ALC) Protocol Instantiation", RFC 3450, December 2002.

[3] Luby, M., Gemmell, J., Vicisano, L., Rizzo, L., Handley, M. and J. Crowcroft, "Layered Coding Transport (LCT) Building Block", RFC 3451, December 2002.

[4] Luby, M., Gemmell, J., Vicisano, L., Rizzo, L., Handley, M. and J. Crowcroft, "Forward Error Correction (FEC) Building Block", RFC 3452, December 2002.

[5] Mills, D., "Network Time Protocol (Version 3), Specification,

Implementation and Analysis", RFC 1305, March 1992.

[6] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[7] Luby, M. and L. Vicisano, "Compact Forward Error Correction (FEC) Schemes", draft-ietf-rmt-bb-fec-supp-compact-01 (work in progress), May 2003.

[8] Thompson, H., Beech, D., Maloney, M. and N. Mendelsohn, "XML Schema Part 1: Structures", W3C Recommendation, May 2001.

[9] Biron, P. and A. Malhotra, "XML Schema Part 2: Datatypes", W3C Recommendation, May 2001.

Informative references

[10] Deutsch, P. and J-L. Gailly, "ZLIB Compressed Data Format Specification version 3.3", RFC 1950, May 1996.

[11] Handley, M., Perkins, C. and E. Whelan, "Session Announcement Protocol", RFC 2974, October 2000.

[12] Deering, S., "Host Extensions for IP Multicasting", RFC 1112, STD 5, August 1989.

[13] Perrig, A., Canetti, R., Song, D. and J. Tygar, "Efficient and Secure Source Authentication for Multicast, Network and Distributed System Security Symposium, NDSS 2001, pp. 35-46.", February 2001.

[14] Holbrook, H., "A Channel Model for Multicast, Ph.D. Dissertation, Stanford University, Department of Computer Science, Stanford, California", August 2001.

[15] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, May 1996.

[16] Deutsch, P., "GZIP file format specification version 4.3", RFC 1952, May 1996.

[17] Ramsdell, B., "S/MIME Version 3 Message Specification", RFC 2633, June 1999.

[18] Eastlake, D., Reagle, J. and D. Solo, "(Extensible Markup Language) XML-Signature Syntax and Processing", RFC 3275, March 2002.

Authors' Addresses

Toni Paila
Nokia
Itamerenkatu 11-13
Helsinki FIN-00180
Finland

EMail: toni.paila@nokia.com

Michael Luby
Digital Fountain
39141 Civic Center Dr.
Suite 300
Fremont, CA 94538
USA

EMail: luby@digitalfountain.com

Rami Lehtonen
TeliaSonera

Hatanpaan valtatie 18
Tampere FIN-33100
Finland

EMail: rami.lehtonen@teliasonera.com

Vincent Roca
INRIA Rhone-Alpes
655, av. de l'Europe
Montbonnot
St Ismier cedex 38334
France

EMail: vincent.roca@inrialpes.fr

Rod Walsh
Nokia
Visiokatu 1
Tampere FIN-33720
Finland

EMail: rod.walsh@nokia.com

Appendix A. Receiver operation (informative)

This section gives an example how the receiver of the file delivery session may operate. Instead of a detailed state-by-state specification the following should be interpreted as a rough sequence of an envisioned file delivery receiver.

1. The receiver obtains the description of the file delivery session identified by the pair: (source IP address, Transport Session Identifier). The receiver also obtains the destination IP addresses and respective ports associated with the file delivery session.

2. The receiver joins the channels in order to receive packets associated with the file delivery session. The receiver may schedule this join operation utilizing the timing information contained in a possible description of the file delivery session.

3. The receiver receives ALC/LCT packets associated with the file delivery session. The receiver checks that the packets match the declared Transport Session Identifier. If not, packets are silently discarded.

4. While receiving, the receiver demultiplexes packets based on their TOI and stores the relevant packet information in an appropriate area for recovery of the corresponding file. Multiple files can be reconstructed concurrently.

5. Receiver recovers an object. An object can be recovered when an appropriate set of packets containing Encoding Symbols for the transport object have been received. An appropriate set of packets

is dependent on the properties of the FEC Encoding ID and FEC Instance ID, and on other information contained in the FEC Object Transmission Information.

6. If the recovered object was an FDT Instance with FDT Instance ID 'N', the receiver parses the payload of the instance 'N' of FDT and updates its FDT database accordingly. The receiver identifies FDT Instances within a file delivery session by the EXT_FDT header extension. Any object that is delivered using EXT_FDT header extension is an FDT Instance, uniquely identified by the FDT Instance ID. Note that TOI '0' is exclusively reserved for FDT delivery.

7. If the object recovered is not an FDT Instance but a file, the receiver looks up its FDT database to get the properties described in the database, and assigns file with the given properties. The receiver also checks that received content length matches with the

description in the database. Optionally, if MD5 checksum has been used, the receiver checks that calculated MD5 matches with the description in the FDT database.

8. The actions the receiver takes with imperfectly received files (missing data, mismatching digestive, etc.) is outside the scope of this specification. When a file is recovered before the associated file description entry is available, a possible behavior is to wait until an FDT Instance is received that includes the missing properties.

9. If the file delivery session end time has not been reached go back to 3. Otherwise end.

Appendix B. Example of FDT Instance (informative)

```
<?xml version="1.0" encoding="UTF-8"?>
<FDT-Instance xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:fl="http://www.example.com/flute"
xsi:schemaLocation="http://www.example.com/flute-fdt.xsd"
Expires="2890842807">
    <File
        Content-Location="www.example.com/menu/tracklist.html"
        TOI="1"
      Content-Type="text/html"/>
    <File
        Content-Location="www.example.com/tracks/track1.mp3"
        TOI="2"
      Content-Length="6100"
      Content-Type="audio/mp3"
      Content-Encoding="gzip"
      Content-MD5="Eth76GlkJU45sghK"
      Some-Private-Extension-Tag="abc123"/>
</FDT-Instance>
```

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

Full Copyright Statement

Copyright (C) The Internet Society (2003). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assignees.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING

BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION

HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.